Универзитет у Београду

Електротехнички факултет



Детекција значајних тачака на лицу уз помоћ алгоритама машинског учења

Софтверско инжењерство великих база података

|  |  |
| --- | --- |
| Ментор: | Студент: |
| др Мирослав Бојовић  Стефан Тубић | Матија Додовић,  2022/3053 |

Београд, јануар 2023.

Садржај

[Садржај i](#_Toc111709988)

[1. Увод 1](#_Toc111709989)

[2. Опис модела 4](#_Toc111709990)

[2.1. Нормализована шема 4](#_Toc111709991)

[2.2. Комплетно денормализована шема 6](#_Toc111709992)

[2.3. Парцијално денормализована табела 8](#_Toc111709993)

[3. Анализа перформанси NT, FDT и PDT 11](#_Toc111709994)

[3.1. Анализа расподела трансакција 12](#_Toc111709995)

[3.2. Основна анализа над MySQL базом 13](#_Toc111709996)

[3.3. Додатна анализа над MySQL базом 15](#_Toc111709997)

[3.4. Aнализа над SQL Server базом 17](#_Toc111709998)

[4. Закључак 20](#_Toc111709999)

[Литература 21](#_Toc111710000)

[Списак скраћеница 24](#_Toc111710001)

[Списак слика 25](#_Toc111710002)

[Списак табела 26](#_Toc111710003)

[A. Додатак A: Упити за учитавање података у DT 27](#_Toc111710004)

1. Увод

Релациони системи за управљање базама података (*relational database management systems – RDBMSs*) су се развили 1970-тих година као научно-истраживачки пројекти. Први прототипи су били System R [1] и INGRES [2]. Од тада па до данас, *RDBMS*-ови се користе у серверском делу велике већине апликација оријентисаних ка подацима (*data intensive/centric*) у којима је потребна систематична и ефикасна манипулација подацима, као и њихово трајно чување. Ове апликације користе различите програмске интерфејсе, попут ODBC [3], JDBC [4], ADO.NET [5], OLE DB [6] итд. Ови интерфејси енкапсулирају саму базу и поједностављују интеракцију с њом.

Ови системи, у најопштијем случају, испољавају карактеристике једног од два режима рада: *OLAP* (*online analytical processing*), који је махом служио интерној употреби за анализу података и доношење пословних одлука, и *OLTP* (*online transaction processing*), који је са друге стране служио крајњим корисницима система). *OLAP* системи су се заснивали на извршавању, најчешће врло сложених, аналитичких упита, где је најчешћа операција управо операција читања. Подршка за ове системе огледала се у постојању техника за убрзавање сложених аналитичких упита као што су индекси, материјализовани погледи и оптимизатори за звездасте шеме[[1]](#footnote-1). Са друге стране, *OLTP* системи имају интензивно испреплетане операције читања и уписа и у бити су то трансакциони упити. Основна својства ових система су одржавање индекса (коришћењем B-стабла или различитих подваријанти као што су B+, B\* итд.), коришћење оптимизатора упита заснованих на *cost-based* техници и обезбеђивању *ACID* (*atomicity*, *consistency*, *isolation* и *durability*) својства сваке трансакције. Историјски, први *DBMS*-ови су обједињавали оба ова система, што је била такозвана политика *“one size fits all”* [7][8]. Такав систем се испоставио веома компликованим и неодрживим [8], тако да се данас DBMS-ови праве или као *OLTP* или као *OLAP*. Тема овог рада је анализа перформанси денормализованих табела код *OLTP* системима.

Шеме релационе базе података за *OLTP* апликације по правилу су дизајниране тако да поштују принципе нормализације. То подразумева да су модели података направљени тако да у што већој мери смање редудансу у подацима, како би се смањио меморијски простор, осигурала конзистентност података и повећала брзина операција ажурирања података. Приступ дизајнирању модела је најчешће тај да се крене од концептуалног модела, да би се на основу њега направио физички. Дизајн концептуалног модела се може радити у неком од апстрактних језика, попут *UML*-а или Ентитети-везе (*Entity-Relationship*, ER). Погодност овога је та што постоје готови алати који након дизајна преводе концептуални модел у релациони, на пример [9].

Често коришћена ствар у *OLTP* апликацијама које користе нормализоване базе података јесте и *ORM* (*object-relational mapper*), као што су *Entity Framework* [10] и *Hibernate* [11]. Модерни ORM обезбеђује мапирање ентитета (најчешће табеле) директно у објекте објектно оријентисаног програмског језика. Најчешће је то мапирање изведено врло директно из релационог модела тако да колоне из табела одговарају пољима објекта. Проблем који је постојао у ранијим *ORM*-овима се заснива на референцијалним интегритетима. Долазило је до генерисања упита који дохватају податке за сваку појединачну везу страним кључем [12], што је касније исправљено раним учитавањем (*data prefetching*) у току самог извршавања апликације [13]. Тако је број захтева за податке ка бази смањен довлачењем свих потребних података за све везе. Оптимизатори у овим апликацијама везе преко страних кључева најчешће преводе у одговарајућа спајања (*join*). Оптимизатор након тога модификује упит да би операција била што ефикаснија. Осим тога, показује се да *Hibernate-*ов оптимизатор не може у свакој ситуацији да пронађе адекватан упит [14], што онда доводи до проблема у перформансама. Показује се и да оптимизатори самог *DBMS*-a често одступају од оптималних планова извршавања када упит садржи неколико спајања (већ са 3 или 4 се примећују разлике у перформансама) [15].

Начин смештања података у меморији се испоставља као значајан фактор у операцијама читања. Код тзв. система оријентисаних на врсте (*row-oriented* или *row-store*), каква је већина система који се користе у OLTP апликацијама, један слог (запис) табеле смешта се континуално, тако да су различита поља истог записа смештена једно до другог у меморији. То за последицу има значајно мању цену приступа подацима из исте табеле него из различитих табела [16][17][18]. Перформансе читања података из табела са оваквим начином смештања података могу деградирати и услед промашаја у кешевима [19] или услед самих I/O операција на *HDD* (*hard disk drive*) [20] или на *SSD* (*solid state drive*) уређајима [21][22]. У случају да су табеле нормализоване, дешава се да су логички блиски подаци раздвојени у неколико табела. Техника нормализације не води рачуна о захтевима за подацима који се дешавају у току самог извршавања [23], на пример таквим да се одређене групе колона најчешће читају у истој трансакцији. То за последицу има да трансакција која дохвата одређене податке који се логички дохватају заједно мора да извршава спајање табела, што је већ раније поменуто као фактор који значајно успорава целу операцију.

У данашњим системима присутна је и техника дистрибуираног чувања база података. Она се састоји у вертикалној (по колонама) и/или хоризонталној (по редовима) подели табела и смештању таквих делова на појединачне сервере, који се у случају ових система називају чворови (*nodes*). Техника се назива партиционисање и за циљеве има побољшање перформанси приступа подацима због могуће паралелизације и повећање капацитета саме базе података, због могућности додавања, произвољно много, чворова са подацима. Перформансе се, у основи, побољшавају паралелним извршавањем различитих упита, од којих се сваки односи на различиту, и то само једну, партицију. Други, мада ређи, случај паралелизације се може постићи уколико су потребни подаци за упит распоређени по различитим чворовима, али тако да сваки сервер, у паралели, може да достави парцијални резултат, од који се на крају формира коначан резултат. Унапређивање техника убрзавања рада дистрибуираних база података су предмет разних истраживања, на пример [24][25][26][27].

Један од начина убрзања извршавања упита су и материјализовани погледи [28]. Испоставља се да су материјализовани погледи веома ефикасни у *OLAP* системима, зато што обједињују физички раздвојене колоне које су потребне у истом упиту, које се затим смештају у привремену табелу и индексирају ради бржег приступа. Упит који се може испунити из креираног материјализованог погледа се своди на читање из једне табеле, што је веома брза операција. Такође, материјализовани погледи се могу користити и као међурезултати сложенијег упита, било експлицитно од стране програмера, било имплицитно уколико сам планер препозна могућност. У свим наведеним случајевима, операција читања је веома брза. Са друге стране, материјализовани погледи имају мањкавост у одржавању конзистентности услед операција ажурирања. Постоје разне технике од којих је инкрементално ажурирање најисплативија са становишта перформанси [29][30]. Ипак, у модерним *DBMS*-овима су материјализовани погледи ограничено подржани (не постоје у *MySQL*-у, инкрементално одржавање не постоји у *PostgreSQL*-у, *left outer join* се не може наћи у материјализованом погледу у *SQL Server*-у [31]).

Денормализација нормализоване базе је један од приступа који могу побољшати перформансе операција читања. Ове технике су у употреби, мада не формално, у свим системима, а примењују се тако што програмери и администратори база података, на основу искустава, доносе ад-хок одлуке у променама модела [23][32][33]. Ове одлуке се доносе несистематично и зависе од конкретне апликације и коришћења података из саме базе. Додатни проблеми се јављају уколико се, што је и најчешћи случај, користи неки *ORM*, где је потребно заобилазити погодности самог *ORM*-а и ручно имплементирати упите који су временски захтевни. Овакав начин модификовања кода се назива *anti-pattern* [34]. Да би се предупредило овакво дизајнирање и ад-хок мењање база, јављају се систематични приступи денормализацијама [31][35], са крајњим циљем да се цео систем аутоматизује и да се нормализована база денормализује тако да се извуку најбоље перформансе.

Циљ овог рада је да се покаже степен унапређења перформанси операција читања и деградације операције уписа денормализованих шема у односу на нормализоване. Остатак рада је организован на следећи начин: поглавље 2 (Опис модела) представља моделе који ће бити коришћени у експериментима. Описан је нормализовани модел, као и два денормализована модела. Такође наведене су и одговарајуће трансакције за операције читања и уписа, чије ће време трајања бити мерено у експериментима. Поглавље 3 (Анализа перформанси) описује поставку и резултате експеримената. Део овог поглавља се бави и анализом трансакција које ће се извршавати у експериментима. Рад се завршава закључком.

1. Опис модела

Услед потребе за стандардизованим мерењем перформанси нормализоване и денормализоване шеме за задати улазни сет података, коришћен је стандардни *TPCE* бенчмарк за OLTP системе [36]. Мерене су разлике у перформансама нормализоване (у наставку NT) и денормализоване шеме, и то за операције читања (*select* упит) и операције ажурирања (*update* упит).

*TPCE* репрезентује систем фирме на берзи и садржи репрезентативан скуп трансакција за вођење целокупног система трговине на берзи једне фирме. У целом бенчмарку постоји 12 трансакција које су различите комплексности и које циљају различите аспекте пословања на берзи. Свака трансакција подељена је на више “оквира“ (*frames*), где сваки одговара скупу операција које морају бити атомично извршене. Оквири се у складу с тим могу посматрати одвојено (само извршавање, времена и количина података се могу одвојено третирати за сваки оквир). Расподела и фреквенција трансакција је одређена емпиријски [36]. Свака трансакција ради над нормализованом релационом шемом, чије су дефиниције дате у спецификацији самог бенчмарка [36].

* 1. Нормализована шема

У експериментима који су извршени у овом раду коришћен је подскуп трансакција и делова трансакција, чији називи следе: *Customer Position – Frame 1* (у наставку T2F1*[[2]](#footnote-2)*), *Market Feed – Frame 1* (у наставку T3F1), *Trade Result – Frame 2* (у наставку T8F2) и *Trade Result – Frame 6* (у наставку T8F6).

Трансакција T2F1 је изабрана као репрезентативна за операцију читања. Репрезентативност се огледа у томе што она спаја више табела, користећи и left outer join и inner join. Денормализоване табеле ће управо бити дизајниране да неутралишу деградацију перформанси насталих овим спајањима [15]. Релевантaн део Т2F1 трансакцијe је у наставку:

NTТ2F1

**select** first max\_acct\_len rows

acct\_id[] = CA\_ID,

cash\_bal[] = CA\_BAL,

assets\_total[] = ifnull((sum(HS\_QTY \* LT\_PRICE)), 0)

from CUSTOMER\_ACCOUNT

left outer join (

HOLDING\_SUMMARY inner join LAST\_TRADE

on LT\_S\_SYMB = HS\_S\_SYMB

) on HS\_CA\_ID = CA\_ID

where

CA\_C\_ID = cust\_id

group by CA\_ID, CA\_BAL

order by 3 asc

Трансакција T2F1 дохвата податке из три табеле: CUSTOMER\_ACCOUNT (у наставку CA), HOLDING\_SUMMARY (у наставку HS) и LAST\_TRADE (у наставку LT), чије су концизне дефиниције у псеудокоду дате у наставку:

**table** **LAST\_TRADE** (

LT\_S\_SYMB char(15) not null,

LT\_DTS datetime not null,

LT\_PRICE decimal(10,2) not null,

LT\_OPEN\_PRICE decimal(10,2) not null,

LT\_VOL bigint not null,

PK(LT\_S\_SYMB)

);

**table** **CUSTOMER\_ACCOUNT** (

CA\_ID bigint not null,

CA\_B\_ID bigint not null,

CA\_C\_ID bigint not null,

CA\_NAME char(50),

CA\_TAX\_ST smallint not null,

CA\_BAL decimal(12,2) not null,

PK (CA\_ID),

INDEX ON (CA\_C\_ID)

);

**table** **HOLDING\_SUMMARY** (

HS\_CA\_ID bigint not null FK CUSTOMER\_ACCOUNT(CA\_ID),

HS\_S\_SYMB char(15) not null,

HS\_QTY int not null,

PK (HS\_CA\_ID, HS\_S\_SYMB)

);

Дефиниције ових табела су прилагођене експериментима који се врше (само наведене трансакције), тако да су избачени референцијални интегритети на табеле које нису у овом редукованом моделу.

Перформансе операције ажурирања су мерене користећи упите које ажурирају табеле које су коришћене у трансакцији T2F1. Релевантан део поменутих трансакција се налази у наставку:

NTТ3F1

**update** LAST\_TRADE

set

LT\_PRICE = price\_quote[i],

LT\_VOL = LT\_VOL + trade\_qty[i],

LT\_DTS = now\_dts

where

LT\_S\_SYMB = symbol[i]

NTT8F2

**update** HOLDING\_SUMMARY

set

HS\_QTY = hs\_qty + trade\_qty

where

HS\_CA\_ID = acct\_id

and HS\_S\_SYMB = symbol

NTT8F6

**update** CUSTOMER\_ACCOUNT

set

CA\_BAL = CA\_BAL + se\_amount

where

CA\_ID = acct\_id

* 1. Комплетно денормализована шема

Извршене су две денормализације нормализованих табела (CA, HS, LT) у циљу смањења утицаја спајања. Прва денормализација је потпуна (у наставку FDT) и она садржи једну денормализовану табелу са подацима из све три табеле коришћене у трансакцији T2F1. Псеудокод дефиниције ове денормализоване табеле је у наставку:

**table** **DTT2T3T8** (

DT\_CA\_ID bigint not null,

DT\_CA\_BAL decimal(12,2) not null,

DT\_HS\_S\_SYMB char(15) not null,

DT\_HS\_QTY int not null,

DT\_CA\_C\_ID bigint not null,

DT\_LT\_PRICE decimal(10,2) not null,

PK (DT\_CA\_ID, DT\_CA\_BAL, DT\_HS\_S\_SYMB),

INDEX ON (DT\_CA\_C\_ID),

INDEX ON (DT\_HS\_S\_SYMB, DT\_CA\_ID)

);

Мотивација за овакву денормализацију лежи у елиминацији оба спајања приликом извршавања трансакције T2F1. Ова трансакција је прилагођена денормализованој шеми, с тим да се коректност трансакције није променила. Преуређена T2F1 трансакција, као и модификоване трансакције ажурирања, T3F1, T8F2 и T8F6 дате су у наставку:

FDTT2F1

**select** first max\_acct\_len rows

acct\_id[] = DT\_CA\_ID,

cash\_bal[] = DT\_CA\_BAL,

assets\_total[] =

ifnull((sum(DT\_HS\_QTY \* DT\_LT\_PRICE)), 0)

from DTT2T3T8

where

DT\_CA\_C\_ID = cust\_id

and DT\_HS\_S\_SYMB IS NOT NULL

group by DT\_CA\_ID, DT\_CA\_BAL

order by 3 asc

FDTТ3F1

**update** DTT2T3T8

set

DT\_LT\_PRICE = price\_quote[i]

where

DT\_HS\_S\_SYMB = symbol[i]

FDTT8F2

**update** DTT2T3T8

set

DT\_HS\_QTY = hs\_qty + trade\_qty

where

DT\_CA\_ID = acct\_id

and DT\_HS\_S\_SYMB = symbol

FDTT8F6

**update** DTT2T3T8

set

DT\_CA\_BAL = DT\_CA\_BAL + se\_amount

where

DT\_CA\_ID = acct\_id

* 1. Парцијално денормализована табела

Табела DTT2T3T8, у оквиру FDT денормализације, настала је тако да се трансакција T2F1 максимално оптимизује, односно да се сви тражени подаци довуку читањем из једне табеле, а не читањем из резултата спајања. У наставку су упити који дају одговарајуће кардиналности по свакој од груписаних табела (прецизније по колонама по којима су табеле спајане). Ред величине табеле DTT2T3T8 над којим су вршене ова испитивања је .

select max(DT.C), avg(DT.C)

from (

select DT\_CA\_C\_ID, count(\*) C

from DTT2T3T8

group by **DT\_CA\_C\_ID**

) DT

Максимална кардиналност је 152, док је просечна кардиналност 50.

select max(DT.C), avg(DT.C)

from (

select DT\_HS\_S\_SYMB, count(\*) C

from DTT2T3T8

group by **DT\_HS\_S\_SYMB**

) DT

Максимална кардиналност је 107, док је просечна кардиналност 72.

select max(DT.C), avg(DT.C)

from (

select DT\_CA\_ID, count(\*) C

from DTT2T3T8

group by **DT\_CA\_ID**

) DT

Максимална кардиналност је 18, док је просечна кардиналност 9.

На основу ових резултата се закључује да је веза која иде преко колоне DT\_CA\_ID у табели DTT2T3T8 најмање кардиналност и на основу тога је одлучено да се на тој колони пресече денормализација. Све колоне које са собом носи денормализација преко DT\_CA\_ID, а то је још колона DT\_CA\_BAL, одлазе у посебну табелу. Како се овај податак ажурира у трансакцији T8F6, назив нове табеле постаје DTT8F6, док остатак колона остаје у главној денормализованој табели која, аналогно, добија назив DTТ2Т3T8F2. DT\_CA\_ID колона представља страни кључ који повезује ове две табеле. Дефиниције у псеудокоду ове две денормализоване табеле су у наставку:

**table DTT2T3T8F2** (

DT\_CA\_ID bigint not null,

DT\_HS\_S\_SYMB char(15) not null,

DT\_HS\_QTY int not null,

DT\_CA\_C\_ID bigint not null,

DT\_LT\_PRICE decimal(10,2) not null,

PK (DT\_CA\_ID, DT\_HS\_S\_SYMB),

INDEX ON (DT\_CA\_C\_ID),

INDEX ON (DT\_HS\_S\_SYMB, DT\_CA\_ID)

);

**table DTT8F6** (

DT\_CA\_ID bigint not null FK DTT2T3T8F2(DT\_CA\_ID),

DT\_CA\_BAL decimal(12,2) not null,

PK (DT\_CA\_ID)

);

Преуређене трансакције читања и ажурирања се налазе у наставку:

PDTT2F1

**select** first max\_acct\_len rows

acct\_id[] = DT\_CA\_ID,

cash\_bal[] = DT\_CA\_BAL,

assets\_total[] =

ifnull((sum(DT\_HS\_QTY \* DT\_LT\_PRICE)), 0)

from DTT2T3T8F2

inner join DTT8F6

on DTT2T3T8F2.DT\_CA\_ID = DTT8F6.DT\_CA\_ID

where

DT\_CA\_C\_ID = cust\_id

and DT\_HS\_S\_SYMB IS not null

group by DT\_CA\_ID, DT\_CA\_BAL

order by 3 asc

PDTТ3F1

**update** DTT2T3T8F2

set

DT\_LT\_PRICE = price\_quote[i]

where

DT\_HS\_S\_SYMB = symbol[i]

PDTT8F2

**update** DTT2T3T8F2

set

DT\_HS\_QTY = hs\_qty + trade\_qty

where

DT\_CA\_ID = acct\_id

and DT\_HS\_S\_SYMB = symbol

PDTT8F6

**update** DTT8F6

set

DT\_CA\_BAL = DT\_CA\_BAL + se\_amount

where

DT\_CA\_ID = acct\_id

1. Анализа перформанси NT, FDT и PDT

Анализа перформанси NT, FDT и PDT рађена је над две базе података: МySQL и SQL Server. За анализу је коришћен радни скуп од 131187 позива одговарајућих трансакција наведених у претходном поглављу. Ови позиви трансакција су генерисани одговарајућим генератором трансакција доступним уз спецификацију [36].

Подаци за NT табеле су генерисани користећи генератор података доступан уз спецификацију [36]. Број рекорда за сваку табелу је у наставку:

**LAST\_TRADE**: 205 500

**CUSTOMER\_ACCOUNT**: 1 500 000

**HOLDING\_SUMMARY**: 3 780 426

Денормализоване табеле, и FDT и PDT, се попуњавају одговарајућим упитима који су изведени из трансакције T2F1. Начин учитавања података у денормализоване шеме је описан у додатку А. Величине денормализованих табела су дате у наставку:

**DTT2T3T8**: 3 780 426

**DTT2T3T8F2**: 3 780 426

**DTT8F6**: 380 000

Ради ефикаснијег извршавања самог експеримента, погодније је стране кључеве и индексе подићи тек након учитавања података да се не би трошило време на одржавање потребних структура пре краја самог учитавања.

Основни експерименти су извршени користећи *Java* програмски језик уз коришћење JDBC [38][39] за конекцију на базу података. Приликом сваког експеримента подаци су поновно учитани и изнова су подигнути индекси и страни кључеви. Овај део експеримента се назива “хладни старт“ (*cold start*) и није улазио у рачунање времена. Након њега је следило извршавање појединачних трансакција и укупно време трајања свих трансакција улазило је у анализу. Завршна фаза експеримента представљала је срачунавање времена и затварање конекције, и она такође није улазила у временску анализу.

Након извршења трансакција T2F1, FTDT2F1 и PDTT2F1 итерира се кроз цео довучени сет, да би се симулирао рад са подацима и да би се избегао ефекат касног довлачења (биће детаљно описан у делу 3.3). Трансакције које ажурирају податке само су враћале информацију о успешности, што је било довољно за потврду извршења. Режијски трошкови оваквог приступа су били исти за све моделе тако да је то утицало само на апсолутна времена (и то исто за све експерименте), тако да је релативни однос перформанси очуван. Експерименти су понављани више пута и најбоља времена су узета у даље разматрање.

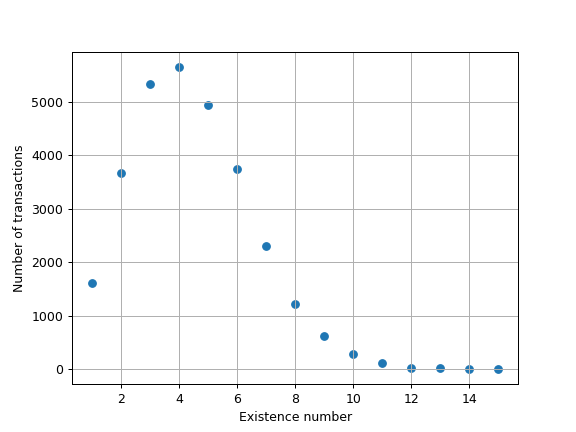
У оквиру *TPCE* спецификације [36] налази се релативна заступљеност свих трансакција у целом бенчмарку. За трансакције од интереса су ти проценти наведени у табели 1. Укупна дужина трајања целог експеримента (оно што се мери је описано у претходном пасусу) ће се анализирати на два начина. Укупна дужина појединачних трансакција може да се упореди између модела, што даје слику о томе колики допринос и колики пенал денормализација уноси одвојено за читање и упис. Други начин је релативно посматрање укупног трајања апликације (сва 4 типа трансакција, пондерисана на проценте из табеле 3.1). Ово поређење даје информацију о томе колико денормализација доприноси целом систему у зависности од карактеристика система.

Табела 3.1. Расподела трансакција у *TPCE*.

|  |  |
| --- | --- |
| **Назив трансакције** | **Заступљеност у *TPCE* [%]** |
| Т2F1 | 13 |
| T3F1 | 1 |
| T8F2 | 10 |
| T8F6 | 10 |

* 1. Анализа расподела трансакција

За трансакцију T2F1, као и свих одговарајућих модификованих трансакција за читање, испитана је расподела параметра *cust\_id*. Одговарајућа расподела је приказана на слици 3.1.



Слика 3.1. Расподела појављивања истих вредности параметра *cust\_id* у скупу трансакција.

Ова расподела говори о томе колико је очекивано истих трансакција у радном скупу, односно колико се различитих вредности CA\_C\_ID користи пошто је то једини параметар у условном (where) делу упита. Расподела показује да највероватнији очекивани број исте вредности параметра *cust\_id* износи 4. Ово разматрање је значајно због могућег ефеката кеширања и учења правилности у самим трансакцијама које у свим експериментима изостаје услед малог броја понављања истих вредности параметра *cust\_id*.

Трансакције које се односе на уписивање (Т3F1, T8F2 и T8F6) су параметризоване тако да одговарају истим рекордима табела које се трансакцијом Т2F1 читају. Ово је урађено да би се очувала изузетно повољна расподела понављања истих трансакција, која је постигнута у случају параметра *cust\_id*.

* 1. Основна анализа над MySQL базом

Експерименти поређења NT, FDT и PDT над MySQL базом података су извршавани над шемама описаним у поглављу 3. Платформа над којом су вршени експерименти је описана у наставку:

*Хардвер*: Лептоп рачунар; AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz; 8 GB RAM; Intel SSD Peknw512G8H 512 GB

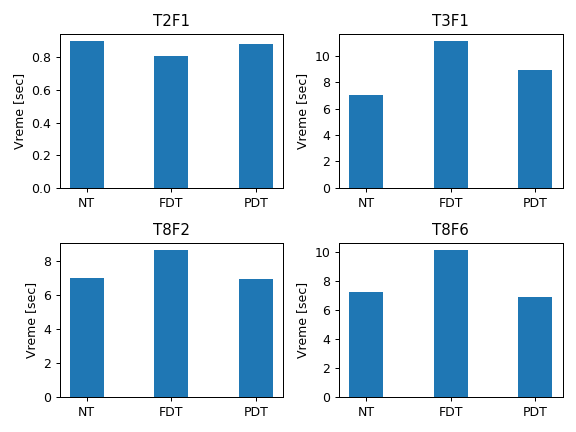
*Оперативни систем*: Windows 10 Home 21H2

*DBMS*: MySQL Ver 8.0.27 for Win64 on x86\_64

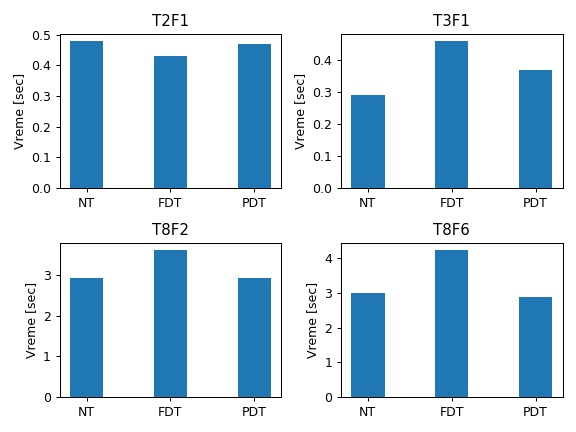
У табели 3.2 су дати резултати експеримента. Колоне означене са ук. представљају укупно време трајања трансакција. Колоне означене са проц. представљају пондерисане вредности укупног трајања трансакција, према фреквенцијама појављивања датим у табели 3.1. На сликама 3.2.1 (укупна времена трајања трансакција) и 3.2.2 (пондерисана времена трајања трансакција) дати су графички прикази резултата експеримента.

Табела 3.2. Резултати основне анализе над MySQL базом.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Трансакција** | **NT**  **ук.**  **[sec]** | **NT**  **проц.**  **[sec]** | **FDT**  **ук.**  **[sec]** | **FDT**  **проц.**  **[sec]** | **PDT**  **ук.**  **[sec]** | **PDT**  **проц.**  **[sec]** |
| Т2F1 | 0.9 | 0.48 | 0.81 | 0.43 | 0.88 | 0.47 |
| T3F1 | 7 | 0.29 | 11.11 | 0.46 | 8.93 | 0.37 |
| T8F2 | 7.03 | 2.93 | 8.65 | 3.6 | 6.98 | 2.91 |
| T8F6 | 7.24 | 3.01 | 10.15 | 4.23 | 6.93 | 2.88 |
| ∑ |  | 6.71 |  | 8.72 |  | 6.63 |



Слика 3.2.1 Апсолутна времена извршавања трансакција над MySQL базом.



Слика 3.2.2 Пондерисана времена извршавања трансакција над MySQL базом.

Апсолутна времена показују унапређење времена извршавања трансакције T2F1. FDTT2F1 има боље време од PDTT2F1 због чињенице да парцијална денормализација има један inner join. У оба случаја су времена извршавања трансакције T2F1 боља од основног времена (NTF2F1). Цена унапређења резултата читања долази за пеналом у трансакцијама уписа, невезано од денормализације. PDT овде има значајно боље резултате од FDT.

У контексту целе апликације, узимајући у обзир заступљеност трансакција у целом *TPCE* набројаних у табели 3.1, FDT има лошији резултат од NT. Кумулативно лошији резултат долази од незанемариве заступљености трансакција уписа, у којима су резултати FDT значајно лошији од NT. Са друге стране, кумулативни резултат PDT је бољи од NT, за шта је заслужан баланс између добити од денормализације у контексту читања и пенала денормализације у контексту уписа.

* 1. Додатна анализа над MySQL базом

Резултати мерења времена извршавања трансакције T2F1 из претходног поглавља су доста блиски за сва три случаја, NT, FDT и PDT. Овакав резултат је последица величине целокупне базе и начина на који MySQL (односно сви потребни подсистеми за извршење упита попут профајлера, оптимизатора итд.) извршавају упите. Да би се избегли наведени утицаји и извукли максимални доприноси денормализације, извршен је експеримент поређења редукованих Т2F1 трансакција.

Да би фокус експеримента био на мерењу утицаја спајања, из трансакција су избачени функција sum, group by и order by. Пошто су ово упити који раде над целим табелама, значајан фактор је и време трансфера података. МySQL база извршава овакве упите тако што се захтев за резултатом изда самој бази да би потом она слала резултате инкрементално, односно извршава сваку итерацију својих петљи на појединачан захтев од итератора којим апликација чита резултате упита. Оваква чињеница нарушава концепт атомичног мерења времена извршавања упита јер је извршавање целог упита подељено на извршавање сваке појединачне итерације кроз резултантни скуп. Тако се у паралели извршава део упита док се део резултата испоручује, и то у неправилним и случајним временским тренуцима без могућности мерења и раздвајања времена извршавања и времена слања резултата, што представља пенал у извршавању [15]. Да би се дужина трајања упита измерила на коректнији начин, резултат редукованог упита је, у истој трансакцији, искоришћен као параметар агрегатне функције count. На овај начин се форсира израчунавање целог резултата упита пре враћања првог (и јединог) слога његовог резултата, да би се након извршавања избројио коначан број редова. Овај број није од интереса, а значајно је што је он скаларан, односно довољно мали да његово слање не утиче на мерење времена. Редуковане трансакције се налазе у наставку:

RNTТ2F1

**select count** (NT.CA\_ID)

from (

select CA\_ID, CA\_BAL

from CUSTOMER\_ACCOUNT

left outer join (

HOLDING\_SUMMARY

inner join LAST\_TRADE

on (LT\_S\_SYMB = HS\_S\_SYMB)

) on HS\_CA\_ID = CA\_ID

) NT

RFDTT2F1

**select count** (FDT.CA\_ID)

from (

select DT\_CA\_ID AS CA\_ID, DT\_CA\_BAL AS CA\_BAL

from DTT2T3T8

where DT\_HS\_S\_SYMB IS NOT NULL

) FDT

RPDTT2F1

**select count** (PDT.CA\_ID)

from (

select DT\_CA\_ID AS CA\_ID, DT\_CA\_BAL AS CA\_BAL

from DTT2T3T8F2

inner join DTT8F6

on DTT2T3T8F2.DT\_CA\_ID = DTT8F6.DT\_CA\_ID

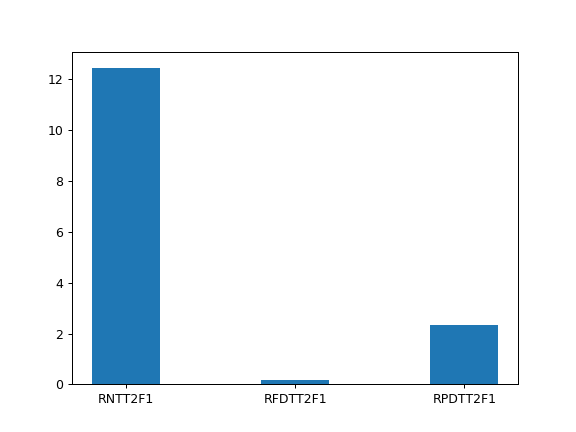
where DT\_HS\_S\_SYMB IS NOT NULL

) PDT

Резултати ових упита дати су у табели 3.3. Графички приказ одговарајућих времена дат је на слици 3.3

Табела 3.3. Резултати додатне анализе над MySQL базом.

|  |  |
| --- | --- |
| **Назив трансакције** | **Дужина трансакције [sec]** |
| RNTТ2F1 | 12.438 |
| RFDTТ2F1 | 0.187 |
| RPDTТ2F1 | 2.328 |



Слика 3.3 Времена извршавања редукованих упита над MySQL базом.

Резултати ове анализе показују значајно веће побољшање перформанси код денормализованих шема у односу на нормализоване. Уклањањем свих утицаја (агрегата и сортирања) сем спајања, показује се да денормализација шеме може значајно побољшати перформансе над упитима код којих су спајања главни узрок деградације перформанси. Додатно се показује значајно веће побољшање перформанси читања код комплетно денормализоване базе у односу на парцијално, које се директно огледа у постојању једног inner join.

* 1. Aнализа над SQL Server базом

Експерименти поређења NT, FDT и PDT над SQL Server базом података су извршавани над шемама описаним у поглављу 2. Платформа над којом су вршени експерименти је описана у наставку:

*Хардвер*: Лептоп рачунар; Intel(R) Core(TM) i7-5500U 2.40 GHz; 8 GB DDR3 RAM; Patriot Burst SSD 512 GB

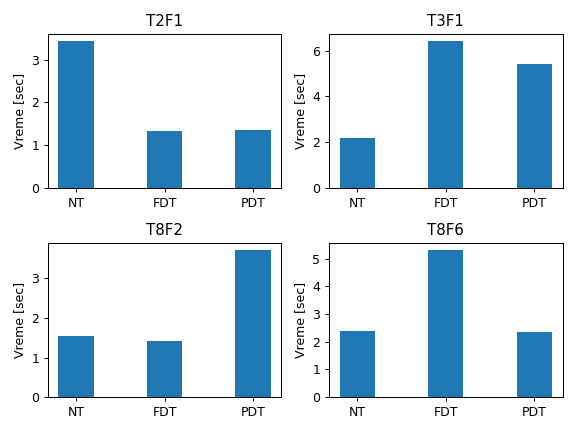
*Оперативни систем*: Windows 10 Enterprise 21H2

*DBMS*: Microsoft SQL Server 2019 (RTM)

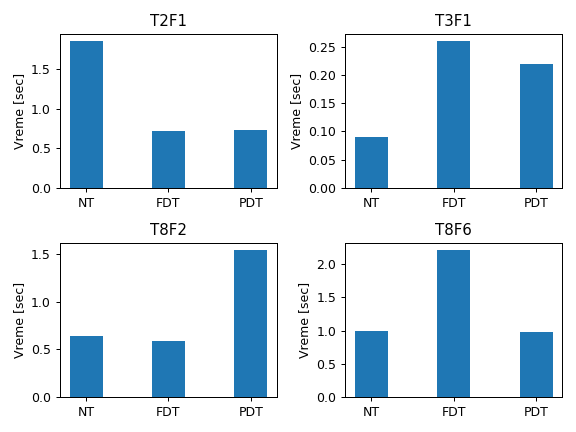
У табели 3.4 су дати резултати експеримента. Колоне означене са ук. представљају укупно време трајања трансакција. Колоне означене са проц. представљају пондерисане вредности укупног трајања трансакција. На сликама 3.4.1 (укупна времена трајања трансакција) и 3.4.2 (пондерисана времена трајања трансакција) дати су графички прикази резултата експеримента.

Табела 3.4. Резултати основне анализе над SQL Server базом.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Трансакција** | **NT**  **ук.**  **[sec]** | **NT**  **проц.**  **[sec]** | **FDT**  **ук.**  **[sec]** | **FDT**  **проц.**  **[sec]** | **PDT**  **ук.**  **[sec]** | **PDT**  **проц.**  **[sec]** |
| Т2F1 | 3.44 | 1.86 | 1.33 | 0.72 | 1.36 | 0.73 |
| T3F1 | 2.2 | 0.09 | 6.42 | 0.26 | 5.41 | 0.22 |
| T8F2 | 1.54 | 0.64 | 1.42 | 0.59 | 3.71 | 1.54 |
| T8F6 | 2.38 | 0.99 | 5.31 | 2.21 | 2.35 | 0.98 |
| ∑ |  | 3.58 |  | 3.78 |  | 3.47 |



Слика 3.4.1 Апсолутна времена извршавања трансакција над SQL Server базом.



Слика 3.4.2 Пондерисана времена извршавања трансакција над SQL Server базом.

Апсолутна времена показују унапређење времена извршавања трансакције T2F1. FDTT2F1 има боље време од PDTT2F1 због чињенице да парцијална денормализација има један inner join. У оба случаја су времена извршавања трансакције T2F1 боља од основног времена (NTF2F1). Цена унапређења резултата читања долази за пеналом у трансакцијама уписа, невезано од денормализације. PDT у тим трансакцијама има боље резултате од FDT. Овде до изражаја долази предност у трансакцији PDTT8F6, која ажурира само малу табелу, за разлику од FDTT8F6 која свакако мора да итерира кроз целу денормализовану табелу да би постигла исти резултат. Закључци везано за резултате експеримента у контексту целе апликације су истоветни као и у експерименту над MySQL (описани у поглављу 3.2).

Разлике између резултата експеримента над базама MySQL и SQL Server постоје. Резултати у експерименту над SQL Server базом показују веће апсолутне разлике у временима извршавања (исти закључак је и за апсолутна и за пондерисана времена) у свим трансакцијама него што је то случај са MySQL базом. У случају трансакције T2F1 над SQL Server, времена извршавања обе денормализоване шеме, и FDT и PDT, су значајно боља него што је то случај са MySQL базом. Са друге стране, разлика и у деградацији перформанси извршавања свих трансакција уписа над MySQL базом је мања него што је случај са SQL Server базом. Овим је показано да, комерцијална, SQL server база може значајније да искористи позитивне стране денормализације, побољшање перформанси операција читања, и самим тим производи већи пенал у случају операција читања.

1. Закључак

У овом раду је представљена анализа перформанси операција уписа и читања код модела базе који је нормализован, као и две варијанте денормализоване шеме. Показано је да постоји добит од денормализације у операцијама читања, који се огледа у смањењу спајања табела (*join*). Такође је потврђено и да постоји незанемариво успорење операција уписа у табеле денормализоване шеме у односу на одговарајући нормализовани модел [35].

Показано је да различити начини денормализације могу значајно да утичу на перформансе система. Адекватан избор денормализације може да побољша перформансе код операција читања, а да са друге стране не поквари значајно перформансе код операција уписа. Перформансе упита, уколико је спроведена оптимална денормализација, би требало да буду блиске перформансама упита нормализоване шеме у случају операција уписа, док не би требало значајно да одступају од перформанси упита потпуно денормализоване шеме у случају операција читања. Овим радом је показано да добар избор парцијалне денормализације не мора нужно да поквари перформансе операције читања.

Утицај денормализације у контексту целе апликације значајно може да зависи од саме природе апликације (који се мери односом учестаности операција читања и ажурирања). Показано је да различите денормализоване шеме могу и побољшати и деградирати перформансе целокупне апликације, и да избор денормализације мора зависити и од природе саме апликације, а не само од упита који се над базом извршавају.

Додатним анализама је утврђено да сам поступак мерења перформанси може зависити од разних фактора система, као и од избора саме базе података и одговарајућих софтверских подршки за ту базу (оптимизатор, профајлер итд.). Предложен је механизам обмотавања упита, чије се перформансе тестирају, неком од агрегатних функција да би се сви бочни ефекти анулирали и време извршавања упита одговарало реалном трајању упита. Показано је да, поред спајања, значајан утицај на време извршавања упита имају и агрегати, груписања и сортирања. Експериментално је показано да уколико се упит састоји од спајања, без осталих фактора деградирања перформанси, денормализација доноси веома значајна побољшања.

Овим радом је демонстрирана и разлика у извршавању истих трансакција над две различите базе: MySQL и SQL Server. SQL Server база, односно њен оптимизатор упита, је боље искористио предности денормализоване шеме и постигнуто је веће убрзање трансакција читања, док је очекивано лошији пенал испољен у лошијим перформансама операција уписа. Узимајући у обзир анализу утицаја агрегата и сортирања на извршавање упита у МySQL бази, долази се до закључка да оптимизатор упита у SQL Server боље излази на крај са пеналима у перформансама које доносе агрегати и сортирања.

Литература

1. M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. Gray, P. P. Griffiths, W. F. King, R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. Wade, and V. Watson, “System R: A Relational Approach to Database Management”, *ACM Transactions on Database Systems*, 1976.
2. M. Stonebraker, E. Wong, P. Kreps, and G. Held, “The Design and Implementation of INGRES”, *ACM Transactions on Database Systems*, 1(3), pp. 189-222, 1976.
3. Microsoft Open Database Connectivity (ODBC) interface [Online]. <https://docs.microsoft.com/en-us/sql/odbc/reference/odbc-programmer-s-reference?view=sql-server-ver16> (посећен дана 13.08.2022.)
4. Java Database Connectivity [Online]. <http://java.sun.com/javase/6/docs/technotes/guides/jdbc/> (посећен дана 13.08.2022.)
5. ADO.NET [Online]. <https://docs.microsoft.com/en-us/sql/connect/ado-net/microsoft-ado-net-sql-server?view=sql-server-ver16> (посећен дана 13.08.2022.)
6. OLE.DB [Online]. <https://docs.microsoft.com/en-us/sql/connect/oledb/download-oledb-driver-for-sql-server?view=sql-server-ver16> (посећен дана 13.08.2022.)
7. D. Jens, and A. Jindal, “Towards a One Size Fits All Database Architecture “, *CIDR*, pp. 195-198. 2011.
8. M. Stonebraker, and U. Çetintemel, “"One size fits all" an idea whose time has come and gone“, *Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker,* pp. 441-462. 2018.
9. D. Milicev, “Model-driven development with executable UML“, *John Wiley & Sons*, 2009.
10. Entity framework [Online]. <https://docs.microsoft.com/en-us/aspnet/entity-framework> (посећен дана 13.08.2022.)
11. Hibernate framework [Online]. <https://hibernate.org/> (посећен дана 13.08.2022.)
12. С. Guéhis, V. Goasdoué-Thion, and А. P. Rigaux, “Speeding-up data-driven applications with program summaries“, *Proceedings of the 2009 International Database Engineering & Applications Symposium ACM*, Calabria, 2009.
13. P. A. Bernstein, S. Pal, and D. Shutt, “Context-based prefetch–an optimization for implementing objects on relations“, *The VLDB Journal - The International Journal on Very Large Databases*, 9(3), pp. 177-189, 2000.
14. FastOQL: Fast Object Queries for Hibernate [Online]. <https://fastoql.com/> (посећен дана 17.08.2022.)
15. V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann, “How good are query optimizers, really?“, *Proceedings of the VLDB Endowment*, 9(3), pp. 204-215, 2015.
16. H. Plattner, “A common database approach for OLTP and OLAP using an in-memory column database“, *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, pp. 1-2, 2009.
17. D. J. Abadi, P. A. Boncz, and S. Harizopoulos, “Column-oriented database systems“, *Proceedings of the VLDB Endowment*, 2(2), pp. 1664-1665, 2009.
18. S. Harizopoulos, V. Liang, D. J. Abadi and S. Madden, “Performance tradeoffs in read-optimized databases“, *Proceedings of the 32nd international conference on Very large databases*, pp. 487-498, 2006.
19. L. Yinan, and J. M. Patel, “Widetable: An accelerator for analytical data processing“, *Proceedings of the VLDB Endowment*, 7(10), pp. 907-918, 2014.
20. A. Sanjay, S. Chaudhuri, A. Das, and V. Narasayya, “Automating layout of relational databases“, *Proceedings 19th International Conference on Data Engineering,* pp. 607-618, 2003.
21. T. Dimitris, S. Harizopoulos, M. A. Shah, J. L. Wiener, and G. Graefe, “Query processing techniques for solid state drives“, *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data,* pp. 59-72, 2009.
22. J. Do, Y.-S. Kee, J. M. Patel, C. Park, K. Park, and D. J. DeWitt, “Query processing on smart ssds: Opportunities and challenges“, *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013.
23. C. E. Dabrowski, D. K. Jefferson, J. V. Carlis, and S. T. March, “Integrating a knowledge-based component into a physical database design system“, *Information & Management*, 17(2), pp. 71-86, 1989.
24. S. Agrawal, N. Vivek, and Y. Beverly, “Integrating vertical and horizontal partitioning into automated physical database design“, *Proceedings of the 2004 ACM SIGMOD international conference on Management of Data*, pp. 359-370, 2014.
25. J. Zhou, B. Nicolas, and L. Wei, “Advanced partitioning techniques for massively distributed computation“, *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pp. 13-24, 2012.
26. R. Nehme, and N. Bruno, “Automated partitioning design in parallel database systems”, *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pp. 1137–1148, 2011.
27. S. Navathe, S. Ceri, G. Wiederhold, and J. Dou, “Vertical partitioning algorithms for database design“, *ACM Transactions on Database Systems (TODS)*, 4, pp. 680-710, 1984.
28. R. Chirkova, and J. Yang, “Materialized views”, *Foundations and Trends in Databases*, 4(4), pp. 295–405, 2012.
29. P. Larson, and J. Zhou, “Efficient maintenance of materialized outer-join views”, *2007 IEEE 23rd International Conference on Data Engineering*, pp. 56–65, 2007.
30. A. Nica, “Incremental maintenance of materialized views with outerjoins”, *Information Systems*, 37(5), pp. 430–442, 2012.
31. D. Milicev, “Hyper-relations: A Model for Denormalization of Transactional Relational Databases“, *IEEE Transactions on Knowledge and Data Engineering*, 2021.
32. S. K. Shin, and G. L. Sanders, “Denormalization strategies for data retrieval from data warehouses“, *Decision Support Systems*, 42(1), pp. 267-282, 2006.
33. A. Boniewicz, P. Wisniewski, and K. Stencel, “On redundant data for faster recursive querying via ORM systems“, *Computer Science and Information Systems (FedCSIS)*, Kraków, 2013.
34. P. Wegrzynowicz, “Performance antipatterns of one to many association in hibernate“, *Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2013.
35. N. M. Kojić, D. S. Milićev, “Equilibrium of Redundancy in Relational Model for Optimized Data Retrieval“, IEEE Transactions on Knowledge and Data Engineering, 32(9), pp. 1707-1721, 2020.
36. TPCE [Online]. Доступно: <https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-e_v1.14.0.pdf> (посећен дана 08.08.2022.)
37. Bello, Randall G., Karl Dias, Alan Downing, James Feenan, Jim Finnerty, William D. Norcott, Harry Sun, Andrew Witkowski, and Mohamed Ziauddin, “Materialized views in Oracle“, *In VLDB*, 98, pp. 24-27, 1998.
38. JDBC for SQL Server [Online]. Доступно: <https://docs.microsoft.com/en-us/sql/connect/jdbc/microsoft-jdbc-driver-for-sql-server?view=sql-server-ver16> (посећен дана 08.08.2022.)
39. JDBC for MySQL [Online]. Доступно: <https://dev.mysql.com/downloads/connector/j/> (посећен дана 08.08.2022.)

Списак скраћеница

|  |  |
| --- | --- |
| DBMS | *Database Management System* |
| OLAP | *Online Analytical Processing* |
| OLTP | *Online Transaction Processing* |
| ACID | *Atomicity Consistency Isolation Durability* |
| ORM | *Object-Relational Mapping* |
| HDD | *Hard Disk Drive* |
| SSD | *Solid State Drive* |
| NT | *Normalized Table* |
| FDT | *Fully Denormalized Table* |
| PDT | *Partially Denormalized Table* |

Списак слика

[Слика 3.1. Расподела појављивања истих вредности параметра *cust\_id* у скупу трансакција. 12](#_Toc111709954)

[Слика 3.2.1 Апсолутна времена извршавања трансакција над MySQL базом. 14](#_Toc111709955)

[Слика 3.2.2 Пондерисана времена извршавања трансакција над MySQL базом. 14](#_Toc111709956)

[Слика 3.3 Времена извршавања редукованих упита над MySQL базом. 16](#_Toc111709957)

[Слика 3.4.1 Апсолутна времена извршавања трансакција над SQL Server базом. 18](#_Toc111709958)

[Слика 3.4.2 Пондерисана времена извршавања трансакција над SQL Server базом. 18](#_Toc111709959)

Списак табела

[Табела 3.1. Расподела трансакција у *TPCE*. 12](#_Toc111709965)

[Табела 3.2. Резултати основне анализе над MySQL базом. 13](#_Toc111709966)

[Табела 3.3. Резултати додатне анализе над MySQL базом. 16](#_Toc111709967)

[Табела 3.4. Резултати основне анализе над SQL Server базом. 17](#_Toc111709968)

A. Додатак A: Упити за учитавање података у DT

У овом прилогу ће бити излистани упити за учитавање података у денормализоване табеле. FDT табела се попуњава директно из нормализованих (које се учитавају користећи *bulk-load*) и тај упит је дат у наставку:

**insert** **into DTT2T3T8**

(DT\_CA\_ID, DT\_CA\_BAL, DT\_HS\_S\_SYMB,

DT\_HS\_QTY, DT\_CA\_C\_ID, DT\_LT\_PRICE)

select CA\_ID, CA\_BAL, HS\_S\_SYMB, HS\_QTY, C\_ID, LT\_PRICE

from CUSTOMER\_ACCOUNT

left outer join HOLDING\_SUMMARY

on (HS\_CA\_ID = CA\_ID),

LAST\_TRADE, CUSTOMER

where LT\_S\_SYMB = HS\_S\_SYMB

and C\_ID = CA\_C\_ID

PDTтабеле се попуњавају, такође из нормализованих, следећим упитима:

**insert into DTT2T3T8F2**

(DT\_CA\_ID, DT\_HS\_S\_SYMB, DT\_HS\_QTY,

DT\_CA\_C\_ID, DT\_LT\_PRICE)

select CA\_ID, HS\_S\_SYMB, HS\_QTY, C\_ID, LT\_PRICE

from CUSTOMER\_ACCOUNT

left outer join HOLDING\_SUMMARY

on (HS\_CA\_ID = CA\_ID),

LAST\_TRADE, CUSTOMER

where LT\_S\_SYMB = HS\_S\_SYMB

AND C\_ID = CA\_C\_ID;

**insert into DTT8F6** (DT\_CA\_ID, DT\_CA\_BAL)

select CA\_ID, CA\_BAL

from CUSTOMER\_ACCOUNT

left outer join HOLDING\_SUMMARY

on (HS\_CA\_ID = CA\_ID),

LAST\_TRADE, CUSTOMER

where LT\_S\_SYMB = HS\_S\_SYMB

AND C\_ID = CA\_C\_ID

group by CA\_ID, CA\_BAL

1. Звездаста – *star schema* се испоставља као најпогоднија за *OLAP* системе јер страним кључевима обједињује цео систем у централне табеле над којима је ефикасно вршити аналитичке упите [3]. [↑](#footnote-ref-1)
2. Ознаке у називу XТYFZ: X – шема, T – трансакција, Y – редни број трансакције у спецификацији, F – део трансакције, Z – редни број дела трансакције. [↑](#footnote-ref-2)